

SkyEye 用户手册

SkyEye-1.3.2

维护者：Michael.Kang

邮件：blackfin.kang@gmail.com

修订记录

[illegible]

Table of Contents

第一章 SkyEye 的介绍和安装.....	5
1.1 介绍.....	5
1.2 安装.....	5
1.2.1 二进制的 rpm 包安装.....	5
1.2.2 源代码安装.....	5
1.3 安装后的目录和文件.....	6
第二章 SkyEye 快速入门.....	7
2.1 运行 SkyEye 的命令行应用程序.....	7
第三章、SkyEye 命令列表.....	10
3.1 调试相关的命令.....	10
3.1.1 break.....	10
3.1.2 list-bp.....	10
3.1.3 show-step.....	10
3.1.4 stepi.....	10
3.1.5 show-step.....	10
3.1.6 x.....	11
3.1.7 disassemble.....	11
3.1.8 info registers	11
3.2 显示目标板及运行环境.....	12
3.2.1 list-options.....	12
3.2.2 show-map.....	12
3.2.3 show-pref.....	13
3.2.4 list-modules.....	13
3.2.5 list-machines.....	14
3.3 目标板控制命令.....	14
3.3.1 start.....	14
3.3.2 run.....	15
3.3.3 continue.....	16
3.3.4 stop.....	16
3.3.5 quit.....	16
3.4 其它杂项命令.....	16
3.4.1 help.....	16
第四章 SkyEye 配置文件.....	18
4.1、skyeye 的配置文件 skyeye.conf.....	18
4.2、skyeye.conf 文件的格式.....	19
4.3、skyeye.conf 中的不同选项.....	20
4.3.1 arch 选项.....	20
4.3.2 cpu 选项.....	20
4.3.3 mach 选项.....	20
4.3.4 mem_bank 选项.....	20
4.3.5 网络选项.....	21
4.3.6 lcd 选项.....	22
4.3.7 uart 选项.....	22

4.3.8 load_addr 选项.....	22
4.3.9 load_file 选项.....	23
4.3.10 uart_16550.....	23
第五章、在 SkyEye 上运行和调试不同的程序（暂无）	25
第六章、代码覆盖率分析.....	26
6.1、代码覆盖率介绍.....	26
6.2、配置文件选项.....	26
6.3 代码覆盖率数据文件.....	26
6.4 利用 SkyEye 进行代码覆盖率统计.....	27
6.4.1 添加配置选项.....	27
6.4.2 生成数据文件.....	27
6.4.3 数据文件转换.....	27
6.4.4 测试结果验证.....	27
第八章、性能分析模块.....	29
8.1 介绍.....	29
8.2 使用.....	29
第九章 gdb 远程调试.....	30
9.1 介绍	30
9.2 用法	30
9.2.1 在 SkyEye 这一端的操作	30
9.2.2 编译对应平台的 gdb	30
9.2.3 调试程序	30
9.2.4 演示截图.....	31

第一章 SkyEye 的介绍和安装

1.1 介绍

skyeye 是一个支持多架构，多核，并且高度可扩展的硬件仿真平台，当前主要由核心库 libcommon.so 和基于核心库的一系列插件组成。SkyEye 支持的体系结构有 Arm, Blackfin, Coldfire, PowerPC, MIPS, Sparc 和 x86。

SkyEye 的官方网站为 www.skyeye.org，目前由 wangyonghao 负责维护。SkyEye 的源代码通过 svn 仓库进行维护，其地址为：

<http://skyeye.sourceforge.net/>

SkyEye 的源代码当前由 Michael.Kang 负责维护管理，当前 SkyEye 的开发者论坛设为 <http://groups.google.com/group/skyeye-simulator>

可以在这个论坛上讨论 SkyEye 开发中碰到的各种问题和各种想法，SkyEye 的最新版本发布信息也会第一时间发布在上面。

SkyEye 共包含了两个软件包，一个为 skyeye 的发布版本包，另外一个为 skyeye 的测试套件包。一般说来，这两个包会同时发布，并且拥有相同的版本号。测试套件包用来对相应版本的 skyeye 软件包进行测试。

1.2 安装

1.2.1 二进制的 rpm 包安装

对于普通用户来说，可以下载 skyeye 的二进制的 rpm 包进行安装。skyeye 默认的安装路径为 /opt/skyeye。从 SkyEye 官方网站 (<http://sourceforge.net/projects/skyeye/>) 下载 rpm 包，然后进行安装。安装的命令如下，需要用超级用户的权限。

```
rpm -ivh skyeye-1.2.9-rc1.i386.rpm
```

1.2.2 源代码安装

从 sourceforge 网站上下载 Skyeye 的 1.2.9 的源代码，解压

```
tar xzvf skyeye-1.2.9_rc1.tar.gz
```

然后运行如下命令编译

```
./configure  
make lib  
make
```

其中 make lib 来编译第三方的库，make 来编译 skyeye 的源代码

最后运行如下命令安装 SkyEye 到 /opt 目录下

```
make install_lib  
make install
```

1.3 安装后的目录和文件

安装之后，/opt/skyeye 应该有如下目录：

```
bin conf include info lib testsuite
```

其中 bin 目录存放了 skyeye 的二进制程序，描述如下：

mknandflashdump：用来制作 nandflash 的镜像文件

skyeye：skyeye 的命令行应用程序。

prof_convert：用来把代码覆盖率分析的结果文件进行格式转换的程序。

uart_instance：被 skyeye 调用的应用程序，功能为通过一个 xterm 终端来显示串口的输出。

include 目录存放了 skyeye 开发插件时所用到的头文件，这些头文件定义了 skyeye 提供的 API 函数的原型。

info 目录存放了 info 格式的文档，暂时没有提供。

lib 目录存放了 skyeye 的核心库 libcommon.so 和其他一些以动态库存在的插件。

testsuite 目录存放了一个简单的测试用例，用来演示 skyeye 的一些功能。

第二章 SkyEye 快速入门

2.1 运行 SkyEye 的命令行应用程序

直接输入/opt/skyeye/bin/skyeye 会启动 skyeye 的命令行应用程序，并进入 skyeye 的命令行接口，显示如下：

```
ksh@linux-gvai:/opt/skyeye> /opt/skyeye/bin/skyeye

SkyEye is an Open Source project under GPL. All rights of different parts or modules are reserved by
their author. Any modification or redistributions of SkyEye should note remove or modify the
annoucement of SkyEye copyright.

Get more information about it, please visit the homepage http://www.skyeye.org.

Type "help" to get command list.

(skyeye)
```

在 skyeye 的安装目录中包含一个小的 testcase，在目录 testsuite/arm_hello 中。这个测试用例会运行一个不依赖于操作系统的 helloworld，我们可以通过运行这个 helloworld 来学习 skyeye 的基本用法。

运行 skyeye -e arm_hello 命令，进入 skyeye 命令行界面,输出如下：

```
ksh@linux-gvai:/opt/skyeye/testsuite/arm_hello> /opt/skyeye/bin/skyeye -e arm_hello

SkyEye is an Open Source project under GPL. All rights of different parts or modules are reserved by
their author. Any modification or redistributions of SkyEye should note remove or modify the
annoucement of SkyEye copyright.

Get more information about it, please visit the homepage http://www.skyeye.org.

Type "help" to get command list.

(skyeye)
```

然后运行 start 命令加载配置和初始化目标机，命令显示如下：

```
(skyeye)start
arch: arm
cpu info: armv3, arm7tdmi, 41007700, fff8ff00, 0
In do_mach_option, mach info: name at91, mach_init addr 0xb68b2360

uart_mod:3, desc_in:, desc_out:, converter:
In create_uart_console
SKYEYE: use arm7100 mmu ops
exec file "arm_hello"s format is elf32-i386.
load section .text: addr = 0x01000000 size = 0x00000084.
```

```
In tea_write, load_base=0x0,load_mask=0xffffffff
load section .glue_7: addr = 0x01000084 size = 0x00000000.
load section .glue_7t: addr = 0x01000084 size = 0x00000000.
load section .data: addr = 0x01002000 size = 0x00001000.
In tea_write, load_base=0x0,load_mask=0xffffffff
not load section .bss: addr = 0x01003000 size = 0x00000000 .
not load section .debug_abbrev: addr = 0x00000000 size = 0x0000004e .
not load section .debug_info: addr = 0x00000000 size = 0x00000187 .
not load section .debug_line: addr = 0x00000000 size = 0x000000a9 .
not load section .debug_pubnames: addr = 0x00000000 size = 0x0000001c .
not load section .debug_aranges: addr = 0x00000000 size = 0x00000020 .
In SIM_start, Set PC to the address 0x1000000
```

(skyeye)

同时主机上会显示一个 xterm 的窗口，窗口标题为"uart_instance"。截屏如下：

然后我们可以输入"run" 命令来启动 helloworld 的测试用例，输出如下：

我们在 helloworld 运行过程中，也可以随时输入"stop"命令去停止目标板的运行，也可以在运行过程中输入其他命令来显示目标板及其软件的各种信息。

如我们输入"stop"命令来停止目标板，然后输入"info registers" 查看当前的寄存器的数值，如下：

```
(running)stop
(skyeye)info registers
R0    0xf4240
R1    0xffffd001c
R2    0xa
R3    0x58b0e
R4    0x0
R5    0x0
R6    0x0
R7    0x0
R8    0x0
```



```
R9    0x0
R10   0x0
R11   0x1001ffc
R12   0x1000078
R13   0x1001ff0
LR    0x1000018
PC    0x100004c
```

(skyeye)

然后我们可以输入以下命令对当前 PC 之后的指令进行，进行反汇编：

```
(skyeye)disassemble 0x100004c
```

```
cmp    r3, r0
bne
mov    r2, #0 ; 0x0
ldrb   r3, [ip, r2]
add    r2, r2, #1 ; 0x1
cmp    r2, #9 ; 0x9
str    r3, [r1]
ble
b
tsteq  r0, r8, ror r0
```

(skyeye)

第三章、SkyEye 命令列表

3.1 调试相关的命令

3.1.1 break

命令：break [断点地址]

说明：对某一个地址下断点

描述：

用例：

```
(skyeye)break 0x1000050  
Insert breakpoint at address 0x1000050 successfully.
```

3.1.2 list-bp

命令：list-bp

说明：列出当前所有的断点

描述：

用例：

```
(skyeye)list-bp  
ID    Address  
1     0x1000050
```

3.1.3 show-step

命令：show-step .

说明：显示当前运行的指令数目

描述：

用例：

3.1.4 stepi

命令：stepi [number of instruction]

说明：单步运行，可以指定运行多少条指令。

描述：

示例：

```
(skyeye)stepi 100  
In skyeye_stepi, stopped_step=100
```

3.1.5 show-step

命令：stepi [number of instruction]

说明：显示当前运行的指令数目

描述：

示例：

```
(skyeye)show-step  
steps: 5276005
```

3.1.6 x

命令 : x [the address of memory]

说明: 显示某一物理内存地址的值

描述 :

示例 :

```
(skyeye)x 0x129f798
```

```
0x129f798:0xac85fff8
```

3.1.7 disassemble

命令 : disassemble [the address of memory]

说明: 反汇编某一物理内存的值为指令

描述 :

示例 :

```
(skyeye)disassemble 0x129f798
```

```
stcge 15, cr15, [r5], {248}
```

```
stcge 15, cr15, [r5], {252}
```

```
strne pc, [r4, #-4078]!
```

```
andeq r0, r0, r0
```

```
stfccd f0, [r9], {42}
```

```
strcs pc, [r9, #-2044]!
```

```
teqeq r8, r3, lsr #16
```

```
teqeq r0, r8
```

```
addeq r2, r8, r1, lsr #32
```

```
stcge 15, cr15, [r5], {192}
```

```
(skyeye)
```

3.1.8 info registers

命令: infor registers

说明: 显示当前处理器的寄存器的值

描述 :

示例 :

```
(skyeye)info registers
```

```
R0 0x0
```

```
R1 0x81560000
```

```
R2 0x87ec8000
```

```
R3 0x812d0e50
```

```
R4 0x87ec8d00
```

```
R5 0x0
```

```
R6 0x1000
```

```
R7 0x400
```

```
R8 0x0
```

```
R9 0x87ec9000
```

```
R10 0xac220000
```

```
R11 0x8f81801c
```

```
R12 0x0
```

```
R13 0x400
```

```
R14 0xa
```

```
R15 0x24217174
```

```
R16  0x819cf710
R17  0x87ec8000
R18  0x0
R19  0x819cf710
R20  0x87fdfdec
R21  0x400
R22  0x87fae420
R23  0x819cf710
R24  0x0
R25  0x2
R26  0x87febd08
R27  0x87febd08
R28  0x87fea000
R29  0x87febcc8
R30  0x87fae4cc
R31  0x81293584
PC   0x8129f798
(skyeye)
```

3.2 显示目标板及运行环境

3.2.1 list-options

命令：list-options

说明：显示当前 SkyEye 配置文件支持的选项

描述：

用例：.

```
(skyeye)list-options
```

Option Name	Description
nandflash	
cpu	Processor option for arm architecture.
uart	Uart settings
net	Netcard settings
lcd	
mach	machine option
mem_bank	
arch	support different architectures.
cpu	Do not need to provide cpu option any more.

```
(skyeye)
```

3.2.2 show-map

命令：show-map

说明：显示当前的地址分布，以及不同设备所占的地址空间

描述：

示例：

```
(skyeye)show-map
```

Start Addr	Length	Type
0xc0000000	0xc1000000	memory
0xc1000000	0xc1600000	memory
0xc1600000	0xc2000000	memory
0x48000000	0x68000000	IO
0x19000300	0x19000320	IO

(skyeye)

3.2.3 show-pref

命令：show-pref

说明：显示当前的 skyeye 的一些预置的运行选项

描述：这些运行选项可以在 skyeye 命令行启动的时候传给 skyeye，也可以通过 skyeye 的图形窗口进行设置。

示例：

```
(skyeye)show-pref
Module search directories:  /opt/skyeye/lib/skyeye/
Boot address:              0x0
Executable file:           (null)
Load base for executable file: 0x0
Load mask for executable file: 0x0
SkyEye config file:        skyeye.conf
Endian of exec file:       Little endian
(skyeye)
```

3.2.4 list-modules

命令：list-modules

说明：列出所有的已加载的模块名称以及模块的动态库文件

描述：

示例：

```
(skyeye)list-modules
Module Name      File Name
nandflash        /opt/skyeye/lib/skyeye/libnandflash.so
arm              /opt/skyeye/lib/skyeye/libarm.so
log-pc           /opt/skyeye/lib/skyeye/log.so
bfin             /opt/skyeye/lib/skyeye/libbfin.so
log-pc           /opt/skyeye/lib/skyeye/liblog.so
uart             /opt/skyeye/lib/skyeye/libuart.so
disassemble      /opt/skyeye/lib/skyeye/libdisasm.so
mips             /opt/skyeye/lib/skyeye/libmips.so
net              /opt/skyeye/lib/skyeye/libnet.so
code_cov         /opt/skyeye/lib/skyeye/libcodecov.so
sparc            /opt/skyeye/lib/skyeye/libsparc.so
ppc              /opt/skyeye/lib/skyeye/libppc.so
touchscreen      /opt/skyeye/lib/skyeye/libts.so
coldfire         /opt/skyeye/lib/skyeye/libcoldfire.so
flash            /opt/skyeye/lib/skyeye/libflash.so
lcd              /opt/skyeye/lib/skyeye/liblcd.so
```

```
gdbserver /opt/skyeye/lib/skyeye/libgdbserver.so  
(skyeye).
```

3.2.5 list-machines

命令：list-machines

说明：列出当前模拟器支持的处理器类型

描述：

示例：

```
(skyeye)list-machines
```

```
Machine Name
```

```
omap5912
```

```
ps7500
```

```
lpc2210
```

```
ns9750
```

```
sharp_lh7a400
```

```
s3c2440
```

```
s3c2410x
```

```
at91rm92
```

```
pxa_mainstone
```

```
pxa_lubbock
```

```
sa1100
```

```
cs89712
```

```
ep9312
```

```
lh79520
```

```
ep7312
```

```
s3c3410x
```

```
s3c4510b
```

```
lpc
```

```
at91
```

```
(skyeye)
```

3.3 目标板控制命令

3.3.1 start

命令：start

说明：加载配置文件并初始化相应的数据结构

描述：

示例：

```
(skyeye)start
```

```
arch: mips
```

```
Error: Unkonw cpu name "mips"
```

```
"cpu" option parameter error!
```

```
In do_mach_option, mach info: name gs32eb1, mach_init addr 0xb7937d20
```

```

Unkonw option: log
uart_mod:3, desc_in:, desc_out:, converter:
In create_uart_console
In skyeye_read_config, Config format is wrong.

exec file "vmlinux"'s format is elf32-i386.
load section .text: addr = 0x81200000 size = 0x000ad2a0.
In tea_write, load_base=0x0,load_mask=0xffffffff
load section .fixup: addr = 0x812ad2a0 size = 0x0000107c.
In tea_write, load_base=0x0,load_mask=0xffffffff
load section .kstrtab: addr = 0x812ae31c size = 0x0000217c.
In tea_write, load_base=0x0,load_mask=0xffffffff
load section __ex_table: addr = 0x812b04a0 size = 0x00001608.
In tea_write, load_base=0x0,load_mask=0xffffffff
load section __dbe_table: addr = 0x812b1aa8 size = 0x00000000.
load section __ksymtab: addr = 0x812b1aa8 size = 0x000010c8.
In tea_write, load_base=0x0,load_mask=0xffffffff
load section .data.init_task: addr = 0x812b4000 size = 0x00002000.
In tea_write, load_base=0x0,load_mask=0xffffffff
load section .text.init: addr = 0x812b6000 size = 0x000092f8.
In tea_write, load_base=0x0,load_mask=0xffffffff
load section .data.init: addr = 0x812bf2f8 size = 0x000003b0.
In tea_write, load_base=0x0,load_mask=0xffffffff
load section .setup.init: addr = 0x812bf6b0 size = 0x000000a0.
In tea_write, load_base=0x0,load_mask=0xffffffff
load section .initcall.init: addr = 0x812bf750 size = 0x0000004c.
In tea_write, load_base=0x0,load_mask=0xffffffff
load section .data.cacheline_aligned: addr = 0x812c0000 size = 0x00001180.
In tea_write, load_base=0x0,load_mask=0xffffffff
load section .reginfo: addr = 0x812c1180 size = 0x00000018.
In tea_write, load_base=0x0,load_mask=0xffffffff
load section .data: addr = 0x812c2000 size = 0x00290000.
In tea_write, load_base=0x0,load_mask=0xffffffff
not load section .sbss: addr = 0x81552000 size = 0x00000000 .
not load section .bss: addr = 0x81552000 size = 0x0001f4f0 .
not load section .comment: addr = 0x000101f4 size = 0x00000ed6 .
not load section .pdr: addr = 0x00000000 size = 0x00012b60 .
not load section .mdebug.abi32: addr = 0x00000000 size = 0x00000000 .
In SIM_start, Set PC to the address 0x1200464

In gs32eb1_boot_linux, Set PC to the address 0x81200464

(skyeye)

```

3.3.2 run

命令：run

说明：从目标板的 PC 地址开始执行

描述：

示例：

```
(skyeye)run
```

```
(running)
```

3.3.3 continue

命令：continue

说明：在模拟器停止状态下，继续运行

描述：

示例：

```
(skyeye)continue
```

```
(running)
```

3.3.4 stop

命令：stop

说明：在模拟器运行状态下，停止模拟器的运行

描述：

示例：

```
(running)stop
```

```
(skyeye)
```

3.3.5 quit

命令：quit

说明：退出模拟器，整个 skyeye 软件退出

描述：

示例：

```
(skyeye)q
```

```
Destroy threads.
```

```
Unload all modules.
```

```
exit.
```

3.4 其它杂项命令

3.4.1 help

命令：help

说明：获得命令的帮助

描述：

示例：

```
(skyeye)help
```

```
No commands match ". Possibilities are:
```

```
log-pc : record the every pc to log file.
```

```
log-pc : record the every pc to log file.
```

```
disassemble : Disassemble the given address.
```


list-bp : List all the breakpoint.

break : set breakpoint for an address.

show-step : Show the steps of current processor.

x : display memory value at the address.

info : show information for various objects.

load-conf : load a config file and parse it for SkyEye.

list-machines : List all the supported machines for SkyEye.

list-options : List all the available options for SkyEye.

show-map : Show the current memory map for the machine.

show-pref : Show the current preference for SkyEye.

list-modules : List all the loaded module.

start : start simulator.

stepi : step into .

continue : Continue the running of interrupted simulator.

stop : Stop the running of simulator.

run : Start the simulator.

q : Quit SkyEye

quit : Quit SkyEye

ls : Synonym for `list'

? : Synonym for `help'.

help : List all the category for the commands.

help : List all the category for the commands.

(skyeye)

第四章 SkyEye 配置文件

4.1、skyeye 的配置文件 skyeye.conf

skyeye.conf 是 skyeye 的配置文件，用来描述模拟的目标板的类型，内存分布，以及 SkyEye 的运行配置等信息。事实上，你可以把想要让用户配置的选项都可以放在 skyeye 配置文件中。这样用户可以通过编辑 skyeye.conf 文件来比较灵活的选择仿真平台的功能和定制要模拟的目标板。

下面是 skyeye.conf 的一个例子：

```
# skyeye config file for S3C2410X
arch:arm
cpu: arm920t
mach: s3c2410x

# physical memory
mem_bank: map=M, type=RW, addr=0x30000000, size=0x00800000
mem_bank: map=M, type=RW, addr=0x30800000, size=0x00800000, file=./initrd.img
mem_bank: map=M, type=RW, addr=0x31000000, size=0x01000000

# all peripherals I/O mapping area
mem_bank: map=I, type=RW, addr=0x48000000, size=0x20000000

mem_bank: map=I, type=RW, addr=0x19000300, size=0x00000020
net: type=cs8900a, base=0x19000300, size=0x20,int=9, mac=0:4:3:2:1:f, ethmod=tuntap,
hostip=10.0.0.1

lcd: type=s3c2410x, mod=gtk
load_addr:base=0x30000000, mask=0xFFFFF
#dbct:state=on
```

4.2、skyeye.conf 文件的格式

当前，skyeye.conf 中有两种格式的配置选项。

第一种配置选项为简单的配置选项，格式如下：

```
option_name: option_value
```

对于一些如 arch 选项，mach 选项都是这样的格式，例如下面的 arch 选项。

```
arch:arm
```

“arch”是选项的名称，它的值为“arm”。

第二种选项的格式略微复杂，用来设定更多的参数，其格式如下：

```
option_name: arg_name=arg_value, arg_name=arg_value, .....
```

其中 option_name 为选项的名称，arg_name 为选项的参数名称，arg_value 为选项的参数值。举例如下：

```
lcd: type=s3c2410x, mod=gtk
```

“lcd”是选项的名称。“type”是“lcd”这个选项的一个参数，这个参数的值为“s3c2410x”。

而“mod”是“lcd”选项的另外一个参数的名称，它的值为 gtk。

如果配置文件中的一行以“#”开始，则表明这一行为注释。SkyEye 在解析配置文件的时候会忽略这一行。

4.3、skyeye.conf 中的不同选项

当前 SkyEye 支持非常多的选项，通过这些选项我们可以比较容易的去定制和配置我们要模拟的目标硬件。下面我们分别对不同的选项进行比较详细的描述。

4.3.1 arch 选项

选项名称: **arch**

合法的参数值: **arm, blackfin, coldfire, ppc, mips , sparc**

描述：用来指出我们要模拟的体系结构的名称。

示例：

```
arch: arm
```

4.3.2 cpu 选项

选项名称: **cpu**

合法的参数值: SkyEye 支持的不同体系结构支持的处理器系列，例如: arm7dmi, e500 等等。

描述：

示例：

```
cpu: e500
```

4.3.3 mach 选项

选项名称：**mach**

合法的参数值：不同的应用处理器名称，例如：at91, mpc8572 etc.

描述：

示例：

```
mach: at91
```

4.3.4 mem_bank 选项

选项名称: **mem_bank**

描述：用来描述目标机的地址空间分布，例如 IO 空间，内存空间等等。

mem_bank 有很多参数用来描述地址空间的属性，列举如下：

map 参数

参数名称: **map**

合法的参数值：M 代表内存空间，I 代表 IO 空间。

描述：用来指出一段地址空间的属性。

type 参数

参数名称：**type**

合法的参数值：**RW, RO**

描述：选项描述了一段地址空间是可读可写还是只读空间。

addr 参数

参数名称：**addr**

合法的参数值：对于目标处理器的合法地址。

描述：用来指出一段地址空间的起始地址。

size 参数

参数名称：**size**

合法的参数值：一段连续地址空间的大小。

描述：

示例：

```
mem_bank: map=M, type=RW, addr=0x31000000, size=0x01000000
```

上面的选项 mem_bank 描述了一段可读可写的地址空间，类型为内存，起始地址为 0x31000000，它的长度

- 0x1000000。

4.3.5 网络选项

选项名称：net

描述：用来描述目标系统的网卡配置

type 参数

参数名称：**type**

合法的参数值：**cs8900a, rtl8019**

描述：skyeye 模拟的网卡类型

base 参数

参数名称：**base**

合法的参数值：网卡的 IO 空间的起始地址

描述：网卡的 IO 空间的起始地址

size 参数

参数名称：**size**

合法的参数值：网卡 IO 空间的长度

描述：网卡 IO 空间的长度

int 参数

参数名称：**int**

合法的参数值：目标机器分配给网卡的中断号

描述：目标机器分配给网卡的中断号。一般来说，我们可以从硬件的原理图及相关文档上获得。

mac 参数

参数名称：mac

合法的参数值：网卡的 mac 地址

描述：网卡的 mac 地址

ethmod 参数

参数名称：**ethmod**

合法的参数值：**tuntap**

描述：网卡的连接方式。tuntap 是一种点对点的连接方式。

hostip 参数

参数名称：**hostip**

合法的参数值：主机的 ip 地址 the ip address of host machine

描述：主机的 ip 地址，用来和 SkyEye 模拟的目标板进行通讯。对于 tun 模式下的通信，一般来说要和目标板网卡的 IP 地址在同一网段。

4.3.6 lcd 选项

选项名称: lcd

描述: 用来描述 skyeye 模拟的 lcd 控制器的选项。

type 参数

参数名称：**type**

合法的参数值: **s3c2410x, ep7312**

参数描述：支持的 LCD 类型。

mod 参数

参数名称: **mod**

合法的参数值: **gtk**

描述: 用来绘制 LCD 屏幕的底层的 GUI 库。当前我们只使用了 GTK。

4.3.7 uart 选项

选项名称：uart

描述：用来描述和配置目标系统的 uart 硬件

mod 参数

参数名称: **mod**

合法的参数值: **term, stdio, net**

描述: 用来指出当前 uart 仿真使用的底层终端，term 表示另开一个终端窗口做为 uart 的输入和输出，stdio 表示把 uart 的输入和输出混合在 SkyEye 的命令行中，net 表示把 uart 的输入和输出重新定向到某个网络端口中。

4.3.8 load_addr 选项

选项名称：load_addr

描述：用来加载 elf 文件到不同的地址空间

base 参数

参数名称：base

合法的参数值：内存地址

描述：用来指定要加载地址的基址

mask 参数

参数名称：mask

合法参数值：地址掩码，用来和 elf 的加载地址进行或操作，来取出偏移量

描述：mask 需要和 base 来一起使用来加载 elf 文件的所有段，elf 的最终加载地址计算如下公式：

$$\text{真正加载地址} = \text{base} \& (\text{mask} | \text{elf 的加载地址})$$

4.3.9 load_file 选项

选项名称:load_file

描述：用来将文件系统的映像加载到不同的地址空间

filename 参数

参数名称:filename

合法的参数值: 系统中存在的文件路径

描述:用来指定要加载的文件系统映像的文件路径

initrd_start 参数

参数名称:initrd_start

合法参数值:内存地址

描述:用来指定要加载地址的基址

4.3.10 uart_16550

选项名称：uart_16550

描述：用来配置一个 16550 类型的 uart 实例

base 参数

参数名称：base

合法参数值：合法的 IO 地址

描述：uart 外设的 IO 空间的基地址

length 参数

参数名称：length

合法参数值：合法的 IO 地址长度。

描述：uart 的 IO 空间的长度

irq 参数

参数名称：irq

合法参数值：系统中存在的中断号

描述：uart 外设的中断号

配置示例：

uart_16550:base=0xe1000000, length=0x100, irq=0x20

第五章、在 **SkyEye** 上运行和调试不同的程序（暂无）

第六章、代码覆盖率分析

6.1、代码覆盖率介绍

代码覆盖率分析是指统计代码中的执行范围的，有那些地址的代码执行了，哪些地址的代码没有执行。从代码测试的角度去分析，没有被执行到的代码就没有被运行和测试。从一定程度上，代码覆盖率的分析能够评判代码质量。

6.2、配置文件选项

在 SkyEye 中使能代码覆盖率功能，需要在 skyeye.conf 添加如下一行：

```
code_coverage:state=on,start=0x1000000, end=0x1400000, filename=./code_cov
```

其中：code_coverage 为配置选项名称，state 是配置参数

当 state=on，表示使能代码覆盖率；state=off，表示关闭代码覆盖率功能。

start 参数表示开始进行代码覆盖率统计的起始地址。

end: 参数代表进行代码覆盖率统计的结束地址。

filename 参数是指出代码覆盖率所存放的数据文件名称。

6.3 代码覆盖率数据文件

代码覆盖率最终生成的数据文件由两部分组成：文件头和 Profiling 的数据。其中文件头的定义如下：

```
/* The header format of code coverage data file */
#define MAX_DESC_STR 32
typedef struct prof_header_s{
    /* the version of header file */
    int ver;
    /* The length of header */
    int header_length;
    int prof_start;
    int prof_end;
    /* The description info for profiling file */
    char desc[MAX_DESC_STR];
}prof_header_t;
```

6.4 利用 SkyEye 进行代码覆盖率统计

6.4.1 添加配置选项

在配置文件中添加如下行：

```
code_coverage:start=0x1000000, end=0x1400000, filename=./code_cov
```

表示我们统计起始地址为 0x1000000，结束地址为 0x1400000 这个范围的代码的代码覆盖率，并且把结果写入 code_cov 文件。

6.4.2 生成数据文件

然后我们运行 SkyEye 执行目标程序一段时间，会生成一个 code_cov 的数据文件。其中 code_cov 包含了对目标程序的统计信息。

6.4.3 数据文件转换

code_cov 文件是一个二进制的文件，我们在 SkyEye 中还提供了一个工具把代码覆盖率的结果转换为文本的可阅读的格式。其用法如下：

a. 如果不加任何参数，会显示 prof_convert 的用法。示例如下：

```
ksh@server:/opt/testsuite/rtems/build_edb7312_4.9> ./prof_convert
Purpose: print the header info of data file or convert raw data file
to text file.
Usage: ./prof_convert data_file output_file.
Or: ./prof_convert data_file
```

b. 也可以显示代码覆盖率文件的一些格式信息，如下：

```
ksh@server:/opt/testsuite/rtems/build_edb7312_4.9> ./prof_convert code_cov
Version:0x1
Length of header: 0x30 bytes.
From 0x0 to 0x1400000 for code coverage.
Desc: raw data
```

c. 获得可阅读的文本格式，命令示例如下

```
ksh@server:/opt/testsuite/rtems/build_edb7312_4.9> ./prof_convert code_cov /tmp/code_cov.txt
```

运行完这条命令，可以生成一个/tmp/code_cov.txt 的文本文件。

6.4.4 测试结果验证

我们可以打开上面生成的 code_cov.txt 文件，然后再打开我们目标程序的反汇编文件，对代码执行的路径进行验证。下图为我们运行 arm_hello 的测试用例在 SkyEye 中的代码覆盖率。

图的左侧为代码覆盖率的文本输出，图右侧窗口为 arm_hello 的反汇编代码输出。

第八章、性能分析模块

8.1 介绍

SkyEye 的性能分析模块是用来分析 SkyEye 本身的执行速度的模块。关于它的实现细节可以阅读我们的 <<SkyEye Internal>>的相关章节。

8.2 使用

在 SkyEye 运行 start 命令之后，然后运行"pmon"来使能性能监控，然后再运行"run"命令。执行一段程序之后，退出 SkyEye。

在当前的运行目录可以生成一个日志文件 pmon.log，里面记录了 SkyEye 运行的每一秒的 MIPS 值，摘录如下：

```
In 10 seconds, MIPS=6707657
In 11 seconds, MIPS=6696640
In 12 seconds, MIPS=6694161
In 13 seconds, MIPS=6693102
In 14 seconds, MIPS=6683999
In 15 seconds, MIPS=6692131
In 16 seconds, MIPS=6681589
```

第九章 gdb 远程调试

9.1 介绍

gdb remote debugger 模块主要用于对 skyeye 所运行的二进制程序进行远程 gdb 调试。有一点要注意：本地运行的 gdb 为需要配置为对应目标平台的 gdb 程序，往往需要我们自己编译，具体步骤可以看下面编译对应平台的 gdb 的介绍。

9.2 用法

假设我们要调试的二进制程序的文件名称为 vmlinux，然后我们需要进行如下操作。

9.2.1 在 SkyEye 这一端的操作

启动 skyeye，加载二进制程序(elf 格式文件加 -e 文件)，开启 remote-gdb：

```
$ /opt/skyeye/bin/skyeye -e vmlinux
```

```
(skyeye) start
```

```
(skyeye) remote-gdb
```

这时候 skyeye 会开启一个端口号为 12345 的网络服务来和 gdb 进行网络通信。

9.2.2 编译对应平台的 gdb

我们以 arm 平台为例来演示如何编译：

解压 gdb 源码，然后进入 gdb 源码目录，运行 `./configure --target=arm-linux`，然后运行 make，会在 gdb 目录下生成一个 gdb 的文件，

如果你运行这个 gdb 会看到 arm-linux 字样如下：

```
ksh@ksh-server:~/svn/skyeye/skyeye_git/skyeye_1_3_1$ /opt/toolchain/gdb_arm/gdb/gdb
GNU gdb (GDB) 7.0
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-unknown-linux-gnu --target=arm-linux".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb)
```

9.2.3 调试程序

启动二进制程序平台对应编译的 gdb(下面的 gdb 要加路径)，即我们在 2.2.1 编译的 gdb 程序。加载二进制程序，连接 server：

```
$ gdb vmlinux
```

(gdb) target remote:12345 (此步便是与 server 端建立连接)

如果 remote 连接成功，在 server 端将出现 “Remote debugging using host:12345”。之后就可以在 client 端使用 gdb 命令进行调试了，若有打印信息会出现在 server 端。

9.2.4 演示截图

图片说明：左边为 server 端，右边为 client 端。（带有绿色图案的行是用户需要输入的命令，其余为显示部分）

左边 server 端所显示的之前，已经运行了命令行的二进制文件的加载以及 start，当 client 端连接上时，会有第二行的显示。

右边 client 端，在 target remote 远程连接后，就可以使用 gdb 了，我们测试了 breakpoint(break)，以及 continue(c)命令。